Earliest Query Answering for Deterministic Nested Word Automata

Olivier Gauwin[1 2 3], Joachim Niehren[1 3], and Sophie Tison[2 3]

[1] INRIA, Lille
[2] University of Lille 1
[3] Mostrare project, INRIA & LIFL (CNRS UMR8022)

Abstract. Earliest query answering (EQA) is an objective of streaming algorithms for XML query answering, that aim for close to optimal memory management. In this paper, we show that EQA is infeasible even for a small fragment of XPath unless P=NP. We then present an EQA algorithm for queries and schemas defined by deterministic nested word automata (DNWAs) and distinguish a large class of DNWAs for which streaming query answering is feasible in polynomial space and time.

1 Introduction

Streaming algorithms process input streams in an incremental manner, and write their output to some external output collection. The data content on the input stream may be huge, so that only fragments of bounded size can be memorized in main memory at every time point. Furthermore, the input stream is usually restricted to a single reading pass (see [1, 2] for more general models).

Streaming algorithms for XML input data streams that contain XML documents, *i.e.*, linearizations of unranked trees or equivalently nested words. In this paper, we are mainly interested in streaming query answering for node selection queries in XML documents, which return collections of tuples of nodes. Such queries may be defined in the W3C standard language XPath 2.0, whose core has the same expressiveness as first-order logic for unranked trees, or by tree automata. The domain of queries can be restricted by schemas defined by the W3C standard XML Schema or again by tree automata.

For illustration, let us simplify XML documents into words with alphabet $\{a, b\}$, and consider the monadic query Q_0 that selects all b positions succeeded by aa in words of $\{a, b\}^*$. This query can be defined by the first-order formula $lab_b(x) \wedge lab_a(x+1) \wedge lab_a(x+2)$ with one free variable x. Its answer set on word $t_0 = abbaabaaba$ is $Q_0(t_0) = \{3, 6\}$. A streaming algorithm for query Q_0 reads some word $t \in \{a, b\}^*$ from the input stream and computes $Q_0(t)$ incrementally. The first answer candidate encountered is letter b at position 2. Whether it will be selected or not depends on the continuation of the stream, and thus position 2 must be stored by all streaming algorithms. We call such answer candidates *alive*. The next event is letter b at position 3. Good streaming algorithms reject candidate 2 now and discard it from memory. In turn, position 3 becomes alive

and must be stored. It can be safely selected from position 5 on, written to the external output collection, and discarded from internal memory.

We need a notion of streamability that accounts for both space and time. We call a class E of query definitions *streamable*, if there exists a polynomial p and an algorithm mapping query definitions $e \in E$ to streaming algorithms A_e in time $p(|e|)$, such that A_e computes the answer set of query Q_e for all trees t on the input stream, with space and time per step bounded by $p(|e|)$ independently of t. Bar-Yossef et al. [3] showed for a class of XPath queries, that the maximal number of simultaneous alive answer candidates (for all positions of the input stream) is indeed a lower space bound for every streaming query answering algorithm. This number is called the *concurrency* $concur_Q(t)$ of a query Q for an unranked tree t. Classes of queries with unbounded concurrency are thus not streamable. Few positive streamability results exist. Boolean queries (returning true or false) defined by tree automata with languages of trees of bounded depth are streamable [4, 5]. Simply compute all runs in parallel on the fly with a stack of bounded depth. A streaming algorithm with close to optimal memory management for Boolean queries defined in a small fragment of positive Forward XPath was presented in [6]. Benedikt et al. [7] showed P-time streamability for the fragment of Boolean Core XPath 1.0 queries in shallow trees, that never look forwards. Heuristics are proposed to approximate earliest rejection. Streamability results for monadic queries are lacking so far.

Earliest query answering (EQA) is the objective of many recent approaches that hint for streaming algorithms with polynomial time and space [3, 8–11]. The strategy is to memorize alive answer candidates only, in order to reach close to optimal memory management. EQA trades space for time: all EQA algorithms need to decide at every step, whether the current answer candidates are safe for selection or rejection (otherwise they are alive). We call these two decision problems SUFFICIENCY for selection resp. rejection. Benedikt et al. [7] noticed (Theorem 1) that REJECTION SUFFICIENCY for Boolean XPath queries that never look forwards is PSPACE-hard.

As a first contribution, we present hardness results for SELECTION SUFFICIENCY. For arbitrary classes of query definitions, we show how to reduce SELECTION SUFFICIENCY to a language INCLUSION problem. As a corollary, we obtain coNP-hardness of SELECTION SUFFICIENCY for a small fragment of Forward XPath filters with only child and descendant axis (without schemas) by reduction to UNIVERSALITY of Boolean queries [12]. Thus, the P-time streaming algorithms in [8, 13] cannot be earliest, except if P=NP. This result shows that [8] does not fully reach its progressiveness objective, and furthermore it contradicts Theorem 3 of [13] on optimal memory management (without proof). As a counter example, consider the Forward XPath expression //a[not[child::c] and [child::b]], which queries for a-nodes without c-children but with a b-child. Both algorithms will keep a-nodes in memory even when encountering a c-child (and remove them only after closing the last child).

As a second contribution, we provide an EQA algorithm for n-ary queries defined by deterministic nested word automata (DNWAS) [14, 15]. Our result relies

on new automata constructions deciding selection and rejection SUFFICIENCY in an incremental manner. Without determinism, we show that SELECTION SUFFICIENCY for Boolean NWAs is EXPTIME-complete by reduction to UNIVERSALITY of tree automata [16]. Let Q_e be queries with fixed arity $n \geq 0$ that are defined by a pair of dNWAs $e = (A, B)$ which recognizes the canonical language and the domain (aka schema) of the query respectively, and let $t \in L(B)$ be a tree satisfying the schema on the input stream. Our EQA algorithm for e computes $Q_e(t)$ with the following costs, where $d = \mathit{depth}(t)$ is the depth of the tree and $c = \mathit{concur}_{Q_e}(t)$ the concurrency of the query on the tree:

- polynomial precomputation time in $O(|A|^3 \cdot |B|^3)$;
- polynomial space for all steps in $O(c \cdot d \cdot |A| \cdot |B|)$;
- polynomial time for all steps in $O(c \cdot |A|^2 \cdot |B|^2)$.

As a corollary, a subclass E of queries defined by dNWAs is streamable, if depth and concurrency are bounded by some polynomial p such that $d = \mathit{depth}(t) \leq p(|B|)$ and $c = \mathit{concur}_{Q_e}(t) \leq p(|A| \cdot |B|)$ for all $e = (A, B) \in E$ and $t \in L(B)$.

Related work. Preliminary versions of this paper were presented at the workshops Plan-X'08 and AutoMathA'09. Independently, we showed in [17] how to decide bounded concurrency for queries defined by dNWAs in P-time, by reduction to bounded valuedness of recognizable tree relations.

Schemas defined by dNWAs subsume extended deterministic DTDs with restrained competition [18] modulo a P-time transformation. Kumar, Madhusudan and Viswanathan [11] investigate EQA by dNWAs, but for a restricted class of monadic queries which always allow for immediate node selection at opening time. Similarly, Benedikt and Jeffrey [10] consider immediate node selection at opening and closing time, for XPath filters on depth-bounded documents. Madhusudan and Viswanathan [19] propose a streaming algorithm for monadic queries defined by NWAs. They impose a serious restriction on their automata so that the SUFFICIENCY becomes trivially decidable in P-time. This class of automata is not proved to be complete for MSO, and the cost of the translation to this class is not investigated. Our results answer to these questions.

NWAs are equivalent modulo P-time to pushdown forest automata [20, 9, 21]. Berlea's [9] P-time EQA algorithm for queries defined by a variant of pushdown forest automata is very different to ours in that no determinism is assumed. This works out, since he assumes infinite signatures so that UNIVERSALITY and SUFFICIENCY become trivially decidable, in contrast to a finite signature, where SUFFICIENCY becomes EXPTIME-hard. With infinite signature, however, schemas can no more be expressed, closure under complement fails, and MSO is no more captured (in contrast to what is stated in Section 3.1 of [9]). Thus, while EQA becomes much simpler, it looses much of its interest.

Bar-Yossef et al. [3] proposed a streaming algorithm with optimal memory management for monadic queries in shallow trees defined in a fragment of Forward XPath queries. Unfortunately, this algorithm is incorrect since it doesn't try to decide REJECTION SUFFICIENCY. Whether this problem might be solvable by adding further restrictions is open.

Outline. In Sec. 2, we show how to define queries for unranked trees. In Sec. 3 we recall nested word automata. In Sec. 4, we present hardness results for SUFFICIENCY problems. In Sec. 5, we show how to decide SUFFICIENCY for queries defined by dNWAs incrementally, in order to obtain an EQA algorithm in Sec. 6.

Appendix A recalls results on the expressiveness and efficiency of NWAs. Appendix B proves the hardness results for SELECTION SUFFICIENCY. Appendix C proves the correctness of our sufficiency tests for dNWAs, for both selection and rejection. Appendix D presents the details of our EQA algorithm for dNWAs. Appendix E discusses the addition of schemas, also for the case of rejection. Finally, Appendix F presents a example run for our EQA algorithm.

2 Queries and Schemas in Unranked Trees

The set T_Σ of *unranked trees* over a finite set Σ is the least set that contains all pairs $a(t_1, \ldots, t_m)$ consisting of a letter $a \in \Sigma$ and an *hedge* $(t_1, \ldots, t_m) \in T_\Sigma^m$ were $m \geq 0$. Nodes of trees and hedges are words over natural numbers defined by $nod(a(t_1, \ldots, t_m)) = \{\epsilon\} \cup nod((t_1, \ldots, t_m))$ and $nod((t_1, \ldots, t_m)) = \cup_{i=1}^m \{i \cdot \pi \mid \pi \in nod(t_i)\}$. We denote the label of node $\pi \in nod(t)$ by $lab^t(\pi) \in \Sigma$. The empty word ϵ is the root of all trees (but of no hedge). The word $\pi \cdot i \in nod(t)$ is the ith child of π. Let $child^t \subseteq nod(t)^2$ be the father-child relation and $lab^t_a = \{\pi \mid lab^t(\pi) = a\}$ the labeling relation for $a \in \Sigma$.

Linearizations of unranked trees in pre-order are called *nested words* in [15, 14]. We write $\mathrm{nw}(t)$ for the nested word of t. For instance, if $t = a(b, c(d), f)$ then $\mathrm{nw}(t) = (\mathtt{op}, a) \cdot (\mathtt{op}, b) \cdot (\mathtt{cl}, b) \cdot (\mathtt{op}, c) \cdot (\mathtt{op}, d) \cdot (\mathtt{cl}, d) \cdot (\mathtt{cl}, c) \cdot (\mathtt{op}, f) \cdot (\mathtt{cl}, f) \cdot (\mathtt{cl}, a)$. In XML syntax, an opening tag $(\mathtt{op}, a)$ is written as $< a >$ and a closing tag $(\mathtt{cl}, a)$ is written as $< /a >$. We ignore data values throughout this paper. We consider streaming algorithms receiving nested words $\mathrm{nw}(t)$ on the input stream. The positions of $\mathrm{nw}(t)$ can be identified with the following set of events:

$$eve(t) = \{\mathtt{start}\} \cup (\{\mathtt{op}, \mathtt{cl}\} \times nod(t))$$

If $lab^t(\pi) = a$ then event $(\mathtt{op}, \pi)$ specifies an occurrence of opening tag $< a >$ in an XML stream, and event $(\mathtt{cl}, \pi)$ the corresponding occurrence of closing tag $< /a >$. Fig. 1(c) illustrates a nested word with edges of relating corresponding events, established by some parser in parallel preprocessing. Let $\prec^t$ be the total order on $eve(t)$, *i.e.*, the order of the tags in the XML stream, and for every e except `start` let $pr(e)$ be the immediate predecessor of e in that order. For hedges, events are defined by: $eve(h) = \{\mathtt{start}\} \cup (\{\mathtt{op}, \mathtt{cl}\} \times nod(h))$.

A *schema* is a tree language $S \subseteq T_\Sigma$. An n-ary query is a function Q with schema S is a function Q with domain $dom(Q) = S$ that selects a set of n-tuples of nodes $Q(t) \subseteq nod(t)^n$ for every tree $t \in S$. Boolean queries are queries of arity $n = 0$. The canonical language of an n-ary query Q is a set of annotated trees $L_Q \subseteq T_{\Sigma \times \mathbb{B}^n}$, where $\mathbb{B} = \{0, 1\}$ are the Booleans. For all trees $t \in dom(Q)$ and tuples $\nu \in nod(t)^n$, we define an annotated tree $t' = t * \nu$ in $T_{\Sigma \times \mathbb{B}^n}$ that has the same structure as t, *i.e.*, $nod(t') = nod(t)$, while annotating the node labels of t by bit vectors, such that $lab^{t'}(\pi) = (lab^t(\pi), \beta)$ for all nodes $\pi \in nod(t)$, where

$\beta = (b_1, \ldots, b_n)$, $\nu = (\pi_1, \ldots, \pi_n)$ and $b_i = 1 \Leftrightarrow \pi = \pi_i$ for all $1 \leq i \leq n$. The canonical language of an n-ary query Q is the set of all annotated trees for Q, *i.e.*: $L_Q = \{t * \nu \mid \nu \in q(t)\}$. For Boolean queries, $L_Q \subseteq dom(Q)$.

Queries $Q_{(A,B)}$ in unranked trees can be defined by a pair (A, B) of automata with languages $L(A) = L_Q$ and $L(B) = dom(Q_{(A,B)})$. If $L(B) = T_\Sigma$ then we write Q_A for $Q_{(A,B)}$.

3 Nested Word Automata

In the present paper, we consider NWAs [14] as automata operating directly on unranked trees (as proposed in [21] and similarly to pushdown forest automata [20, 9]) whereas they usually operate on linearizations of unranked trees $\mathrm{nw}(t)$ or slightly more general nested words.

An NWA $A = (\Sigma, \Gamma, stat, init, fin, rul)$ consists of a finite signature Σ of node labels, a finite set $stat$ with subsets $init, fin \subseteq stat$ of initial and final states, a finite set Γ of stack symbols, and a set $rul \subseteq \{\mathtt{op}, \mathtt{cl}\} \times \Sigma \times \Gamma \times stat^2$ of rules. We denote rules as:

$$p_0 \xrightarrow{\alpha\ a:\gamma} p_1$$

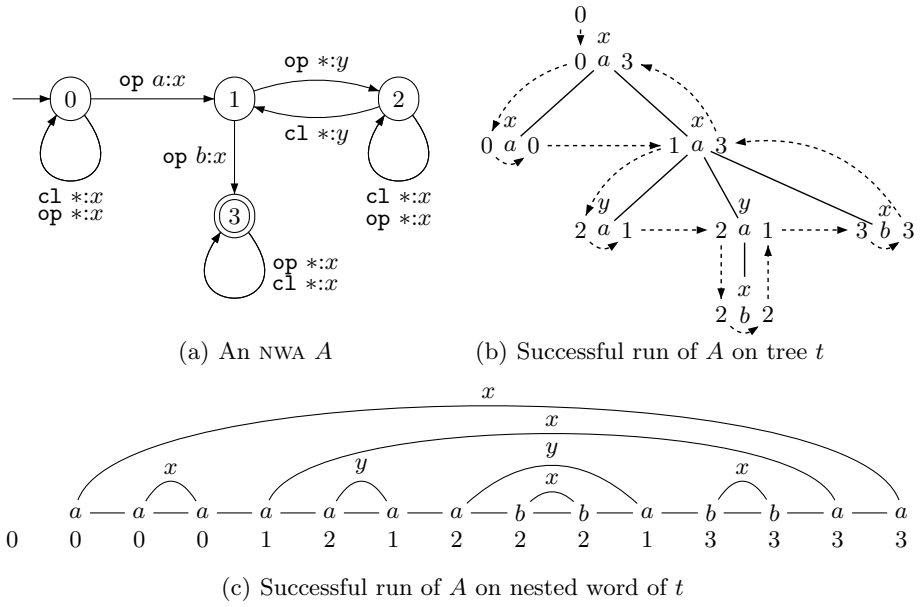
where $\alpha \in \{\mathtt{op}, \mathtt{cl}\}$, $p_0, p_1 \in stat$, $a \in \Sigma$, $\gamma \in \Gamma$. Whenever necessary, we will upper index components of A, as for instance, writing rul^A instead of rul.

An NWA traverses the sequence of events of a given tree t, while annotating all events of t by states and all nodes of t by stack symbols. Let p_0 be the state of the previous event processed, and (α, π) be the current event. The automaton chooses some rule with action α and label $a = lab^t(\pi)$ whose left hand side is p_0. If $\alpha = \mathtt{op}$ then it annotates node π with stack symbol γ. If $\alpha = \mathtt{cl}$ then the rule matches only, if the stack symbol annotated at opening time to π is equal to the stack symbol γ of the rule. For matching rules, the automaton annotates state p_1 on the right hand side to the current event.

More formally, a run of an NWA on a tree t is a function r with two types $r : eve(t) \to stat$ and $r : nod(t) \to \Gamma$ which maps events to states and nodes to stack symbols, such that $r(\mathtt{start}) \in init$ and the following rule belongs to rul for all $\pi \in nod(t)$ with $a = lab^t(\pi)$, and actions $\alpha \in \{\mathtt{op}, \mathtt{cl}\}$.

$$r(pr(\alpha, \pi)) \xrightarrow{\alpha\ a:r(\pi)} r((\alpha, \pi))$$

An example of a run of an NWA on the tree $a(a, a(a, a(b), b)))$ is given in Fig. 1. It tests whether this tree satisfies the Boolean XPath query $[//a[child::b]]$, or equivalently the first-order formula $\exists x(lab_a(x) \wedge \exists y(child(x, y) \wedge lab_b(y)))$. When opening an a-node in its initial state 0, this NWA guesses whether it matches the a-position of the XPath expression (state 1) or not (state 0). From state 1, it waits while traversing a sequence of states $(2^*1)^*$, until some b-child is opened, before concluding success in state 3. The information of being a child of the a-node opened in state 1 is annotated by stack symbol y, and passed over from the left to the right.



(a) An NWA A (b) Successful run of A on tree t

(c) Successful run of A on nested word of t

Fig. 1. An NWA checking the Boolean XPath filter $[//a[child::b]]$ by successful runs.

A run r of A on a tree t is successful if $r((\mathtt{cl}, \epsilon)) \in \mathit{fin}^A$. The set of all possible runs of the NWA A on the tree t is denoted $\mathit{runs}^A(t)$ and the subset of all successful runs by $\mathit{runs\_succ}^A(t)$. The recognized language $L(A)$ is the set of all trees $t \in T_\Sigma$ that permit a successful run by A, *i.e.*, $L(A) = \{t \in T_\Sigma \mid \mathit{runs\_succ}^A(t) \neq \emptyset\}$. For a hedge $(t_1, \ldots, t_k)$, a run is successful if $r(\mathtt{start}) \in \mathit{init}^A$ and $r((\mathtt{cl}, k)) \in \mathit{fin}^A$.

An NWA is *deterministic* or a *d*NWA , if it has a single initial state, no two `op` rules for the same letter use the same state on the left, and no two `cl` rules for the same letter use the same stack symbol and the same state on the left. The unique run of a dNWA A on a tree t can be computed in a streaming manner, if it exists. The input is the nested word $\mathrm{nw}(t)$ of some t that is enriched by the nesting relation by parallel preprocessing with a SAX parser, and the output is the sequence of states that A assigns to the events of t. In order to do so, A has to be stored, and at every time point the current state and the stack of symbols that are annotated to the nodes on the path from the root to the current node. The maximal memory needed at any time point is $O(|A| + \mathit{depth}(t) + 1)$.

4 Complexity of Earliest Query Answering

We present the decision problems of EQA algorithms for n-ary node selection queries and establish lower complexity bounds.

We have to define sufficient events for tuple selection. For every $\eta \in \mathit{eve}(t) - \{\mathtt{start}\}$, let the tree prefix $t^{\preceq \eta}$ be the fragment of t which contains all nodes of

t opened before or at event η. Note that $t^{\preceq(\mathtt{cl},\pi)}$ contains all proper descendants of π in t, while $t^{\preceq(\mathtt{op},\pi)}$ does not. For two trees $t, t' \in T_\Sigma$ and $\eta \in \mathit{eve}(t)$ we define $\mathit{equal}_\eta(t,t')$ by $\eta \in \mathit{eve}(t) \cap \mathit{eve}(t')$ and $t^{\preceq\eta} = t'^{\preceq\eta}$, *i.e.*, t and t' have the same prefix until η.

Definition 1. *(Sufficient events for selection) Let Q be an n-ary query over Σ and $t \in \mathit{dom}(Q)$ a tree. We relate tuples $\nu \in \mathit{nod}(t)^n$ to events $\eta \in \mathit{eve}(t)$ that are sufficient for their selection:*

$$(\nu,\eta) \in \mathit{sel}_Q(t) \;\Leftrightarrow\; (\nu \in \mathit{nod}(t^{\preceq\eta})^n \;\wedge\; \forall t' \in \mathit{dom}(Q).\ \mathit{equal}_\eta(t,t') \Rightarrow \nu \in Q(t'))$$

Note that $(\nu,\eta) \in \mathit{sel}_Q(t)$ implies $\nu \in Q(t)$. Furthermore, successors of sufficient events are sufficient. Consider for instance the monadic query Q_1 with schema $T_{\{a,b,c\}}$ defined by the XPath expression $//a[\mathit{child}::c]/\mathit{child}::b$, or equivalently by the first-order formula $\mathit{lab}_b(x) \wedge \exists y\,(\mathit{lab}_a(y) \wedge \mathit{child}(y,x) \wedge \exists z(\mathit{child}(y,z) \wedge \mathit{lab}_c(z)))$ with one free variable x. On tree $t = b(a, a(a,b,c))$, the earliest time point to select node $2{\cdot}2$ is event $(\mathtt{op}, 2{\cdot}3)$ when the c-child is opened, *i.e.*, $((2{\cdot}2),(\mathtt{op},2{\cdot}3)) \in \mathit{sel}_{Q_1}(t)$. For query Q_2 defined by the same XPath expression, but with a more restrictive schema, requiring that all inner a-nodes have at least one c-child, we can select node $2{\cdot}2$ at opening time, *i.e.*, $((2{\cdot}2),(\mathtt{op},2{\cdot}2)) \in \mathit{sel}_{Q_2}(t)$.

For optimal memory management, it is equally important to discard *rejected* answer candidates in an earliest manner, *i.e.*, candidates that will never be selected in any possible future. Going one step further, one might also want to remove rejected *partial* candidates, for which no completion will ever be selected in any future. Partial candidates ν are elements of $\mathit{nod}_\oslash(t^{\preceq\eta})^n = (\mathit{nod}(t^{\preceq\eta}) \uplus \{\oslash\})^n$, the symbol $\oslash$ denoting components where no selection occurred so far. Completions $\mathit{compl}(\nu,t,\eta)$ are complete candidates obtained by replacing $\oslash$-components of ν by nodes of t opened after η.

Definition 2. *(Sufficient events for rejection) We call a candidate ν* rejected *at event η, or equivalently η sufficient for failing ν, if no completion of ν can be selected in the future:*

$$(\nu,\eta) \in \mathit{rej}_Q(t) \;\Leftrightarrow\; \begin{cases} \nu \in \mathit{nod}_\oslash(t^{\preceq\eta})^n \;\wedge \forall t' \in \mathit{dom}(Q). \\ \mathit{equal}_\eta(t,t') \Rightarrow \forall \nu' \in \mathit{compl}(\nu,t',\eta).\ \nu' \notin Q(t') \end{cases}$$

We call a candidate ν *alive* at event η, if η is not sufficient for selection or rejection of ν, *i.e.*, $(\nu,\eta) \notin \mathit{sel}_Q(t) \cup \mathit{rej}_Q(t)$. EQA algorithms store only alive candidates. The maximal number of alive candidates at a same event is called *concurrency* [3], and written $\mathit{concur}_Q(t)$. For sake of clarity, however, we have to omit the study of REJECTION SUFFICIENCY in the core of the paper (see the appendix).

SUFFICIENCY has to be decided for all candidates by all EQA algorithms at every event. SELECTION SUFFICIENCY for a class of query definitions E is the problem that receives as input a definition $e \in E$ of a query Q_e with arity n, a tree $t \in T_\Sigma$, an event $\eta \in \mathit{eve}(t)$, and a tuple $\nu \in \mathit{nod}(t)^n$, and sends as output

the truth value of $(\nu, \eta) \in sel_Q(t)$. We provide hardness results SELECTION SUFFICIENCY for some query classes E. Proofs and further details, including schemas considerations, are provided in Appendix B.

First, we consider Boolean queries defined by XPath filters F in the following fragment. All trees satisfy the label constraint $*$, while only trees $a(\ldots)$ satisfy the label constraint a. A filter $[\mathit{child}{::}\ell\, F]$ is satisfied by all trees with a subtree at a child position of the root that satisfies ℓ and F. A filter $[//\ell\, F]$ is satisfied by trees having a subtree satisfying ℓ and F. We freely omit filter $[\mathit{true}]$ in examples.

$$F ::= [\mathit{child}{::}\ell\, F] \;\mid\; [//\ell\, F] \;\mid\; [F_1 \;\mathit{and}\; F_2] \;\mid\; [\mathit{not}\, F] \;\mid\; [\mathit{true}] \quad \text{for } \ell \in \Sigma \cup \{*\}$$

Proposition 1. SUFFICIENCY *for Boolean queries defined in the above fragment of XPath is coNP-hard, even without schema assumptions.*

As a consequence, every EQA algorithm for a larger fragment of XPATH cannot be in polynomial time, except if P=NP.

Second, we study this problem for queries defined by automata. For nondeterministic ones, SUFFICIENCY remains hard, even with Boolean queries.

Proposition 2. SUFFICIENCY *for Boolean* NWA *queries is* EXPTIME*-hard.*

However, when restricted to deterministic NWAs, the problem becomes tractable.

Theorem 1. SUFFICIENCY *for n-ary d*NWA *queries is in polynomial time.*

This justifies the use of DNWAs in the following, and shows that SUFFICIENCY is EXPTIME-complete for NWAs (by using NWA determinization).

5 Inferring Safe States

For DNWA queries, we propose a method for deciding SUFFICIENCY at each event of a tree. Our solution is based on a new DNWA construction. Given a DNWA A for the query, we build a DNWA $E(A)$ that accepts the same language, and contains enough information in its states to decide for selection sufficiency at each event immediately. For clarity, sufficiency for rejection is presented in Appendix C, and schemas are discussed at the end of Section 6.

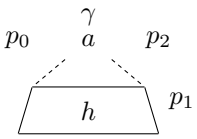
We define a partial run r of an NWA A on a tree t like a run, except that it operates only on a prefix $t^{\preceq\eta}$ for some event $\eta \in eve(t)$. We write $p\_runs^A(t)$ for the set of all partial runs of A on t. Let A be a DNWA over $\Sigma \times \mathbb{B}^n$ defining a query Q_A, $t \in T_\Sigma$, $\eta \in eve(t)$, and $\nu \in nod(t)^n$. We call a state $p \in stat^A$ *safe* for selection of ν at event η if the existence of a partial run r of A on t that maps η to p implies $(\nu, \eta) \in sel_{Q_A}(t)$. In other terms, these are the states that ensure sufficiency for selection when they are reached.

$$\mathit{safe\_sel}^A_{(\nu,\eta)}(t) = \{p \;\mid\; (\exists r \in p\_runs^A(t * \nu) \;\wedge\; r(\eta) = p) \Rightarrow (\nu, \eta) \in sel_{Q_A}(t)\}$$

$$\frac{p_0 \xrightarrow{\texttt{op}\ a:\gamma_1} p_1 \in rul^A \quad \mathcal{S}_1 = univ\_sel^A(a,\gamma_1,\mathcal{S}_0)}{(p_0,\mathcal{S}_0) \xrightarrow{\texttt{op}\ a:(\gamma_1,\mathcal{S}_0)} (p_1,\mathcal{S}_1) \in rul^{E(A)}} \qquad \frac{p_0 \xrightarrow{\texttt{cl}\ a:\gamma_0} p_1 \in rul^A \quad \mathcal{S}_0,\mathcal{S}_1 \subseteq stat^A}{(p_0,\mathcal{S}_0) \xrightarrow{\texttt{cl}\ a:(\gamma_0,\mathcal{S}_1)} (p_1,\mathcal{S}_1) \in rul^{E(A)}}$$

$$init^{E(A)} = (init^A,\ fin^A) \qquad fin^{E(A)} = \{(p,\ fin^A) \mid p \in fin^A\}$$

Fig. 2. Construction of $E(A)$ from A



The remainder of this section describes how these states can be computed by a new dNWA $E(A)$, which permits to decide sufficiency. Here we need some auxiliary definitions. Let $runs^A_{p_0 \to p_1}(h)$ be the set of runs of an NWA A on a hedge h that start in state p_0 and end in state p_1. The operator $ev\_cl^A(h,p_0,a,\gamma)$ evaluates hedge h from state p_0 and subsequently applies a closing rule with label a and state γ:

$$ev\_cl^A(h,p_0,a,\gamma) = \{p_2 \mid \exists r \in runs^A_{p_0 \to p_1}(h).\ p_1 \xrightarrow{\texttt{cl}\ a:\gamma} p_2 \in rul^A\}$$

We consider continuations through hedges in $H_{sel} = T^*_{\Sigma \times \{0\}^n}$. The operator $univ\_sel^A(a,\gamma,P)$ computes all states, from where all hedges in H_{sel} can be evaluated and closed wrt a and γ into a state of $P \subseteq stat^A$:

$$univ\_sel^A(a,\gamma,P) = \{p_0 \mid \forall h \in H_{sel}.\ ev\_cl^A(h,p_0,a,\gamma) \cap P \neq \emptyset\}$$

Given A, t, and ν, we can compute inductively the safe states $S_{sel}(\eta) = safe\_sel^A_{(\nu,\eta)}(t)$ for all events $\eta \in eve(t)$. First, for the closing event of the root, the set of safe states for selection are the final states: $S_{sel}((\texttt{cl},\epsilon)) = fin^A$. Second, at each node π, the safe states for opening events can be computed from those of the corresponding closing event:

$$S_{sel}((\texttt{op},\pi)) = univ\_sel^A(a,\gamma,S_{sel}((\texttt{cl},\pi)))$$

where $a = lab^t(\pi)$ and $\gamma = r^A(\pi)$. Third, the safe states for the opening event of π are equal to those for the closing events of children of π, *i.e.*, $S_{sel}((\texttt{op},\pi)) = S_{sel}((\texttt{cl},\pi{\cdot}i))$.

These propagation rules allow to infer $safe\_sel^A_{(\nu,\eta)}(t)$ for all events η. This can be done by running the NWA $E(A)$ defined in Fig. 2, which adds safe states to each state of A. The signature of $E(A)$ is $\Sigma \times \mathbb{B}^n$ as for A. The state sets may be exponentially large, since $stat^{E(A)} = stat^A \times 2^{stat^A}$ and $\Gamma^{E(A)} = \Gamma^A \times 2^{stat^A}$. Stack symbols are used to pass safe states from parents to all their children.

Proposition 3. *Let A be a* dNWA *on $\Sigma \times \mathbb{B}^n$ that defines a query, and $t * \nu \in T_{\Sigma \times \mathbb{B}^n}$. Then $E(A)$ is a* dNWA *that accepts the same language as A. Furthermore, if r^A (resp. $r^{E(A)}$) is the unique run of A (resp. $E(A)$) on $t * \nu$ then $r^{E(A)}(\eta) = (r^A(\eta), safe\_sel^A_{(\nu,\eta)}(t))$ for all $\eta \in eve(\eta) - \{\texttt{start}\}$.*

$$\frac{a \in \Sigma \times \{0\}^n \qquad p_1 \xrightarrow{\texttt{op}\ a:\gamma} p_3 \in rul^A \qquad p_4 \xrightarrow{\texttt{cl}\ a:\gamma} p_2 \in rul^A}{acc_{H_{sel}}(p_1, p_2) \text{ :- } acc_{H_{sel}}(p_3, p_4).}$$

$$\frac{p \in stat^A}{acc_{H_{sel}}(p, p).} \qquad \frac{p_1, p_2, p_3 \in stat^A}{acc_{H_{sel}}(p_1, p_2) \text{ :- } acc_{H_{sel}}(p_1, p_3), acc_{H_{sel}}(p_3, p_2).}$$

Fig. 3. Inference rules for the definition of $acc^A_{H_{sel}}$

A detailed proof is in Appendix C.[4] Running automaton $E(A)$ for a candidate permits to test sufficiency for selection at the event when it happens. At most one run has to be processed per candidate, thanks to determinism.

6 EQA Algorithm for dNWAs

We present an EQA algorithm for queries defined by dNWAs A which runs in polynomial time per step. The idea is to run the earliest automaton $E(A)$ of Section 5 on the input stream in order to decide SELECTION SUFFICIENCY for all answer candidates at all time points, without constructing $E(A)$ explicitly, since it may be of exponential size compared to A. Recall that deciding REJECTION SUFFICIENCY is needed for all EQA algorithms too. This is explained in the appendix.

Running E(A) on the Fly. Given a dNWA A over $\Sigma \times \mathbb{B}^n$ and a tree $t * \nu$ over the same signature, we want to compute a run of $E(A)$ on $t * \nu$ in polynomial time in the size of A. The application of closing rules of $E(A)$ is easy, since it only has to look for a rule of A. Applying opening rules of $E(A)$ is a little more tedious, since we have to compute the set $univ\_sel(a, \gamma, P)$ while given $a \in \Sigma$, $\gamma \in \Gamma^A$, and $P \subseteq stat^A$.

When assuming the completeness of A beside of determinism (which can be ensured in polynomial time), these sets can be computed by reduction to information on accessibility through hedges for A. Given a set $H \subseteq T^*_{\Sigma \times \mathbb{B}^n}$ of hedges, and states $p_1, p_2 \in stat^A$, we define the following accessibility predicate:

$$acc^A_H(p_1, p_2) \quad \Leftrightarrow \quad \exists h \in H.\ runs^A_{p_1 \to p_2}(h) \neq \emptyset$$

We compute it for $H_{sel} = T^*_{\Sigma \times \{0\}^n}$, with the Datalog program in Fig. 3.

Proposition 4. *The collection of values $acc^A_{H_{sel}}(p_1, p_2)$ can be computed in time $O(|\Sigma| \cdot |A|^3)$ for every d*NWA *A.*

To explain the computation of $univ\_sel^A$, we introduce $befClose^A(a, \gamma, P)$, the set of states that lead to a state of P after closing a with γ:

$$befClose^A(a, \gamma, P) \ = \ \{p_0 \mid \exists p_1 \in P.\ p_0 \xrightarrow{\texttt{cl}\ a:\gamma} p_1 \in rul^A\}$$

[4] Note that for sake of clarity, this construction does not hold for earliest selection of () at the `start` event, for Boolean queries. However, this case can be processed easily by considering every possible label of the root.

```
fun answer(e,t) % e ∈ E, t ∈ dom(Q_e)
   let Q = Q_e
   let candidates = set.new(∅)
in
   for η in eve(t) in streaming-order do
      candidates.update(η)
      for ν in candidates do
        if (ν,η) ∈ sel_Q(t)
           then add-output(ν)
                candidates.remove(ν)
        elseif (ν,η) ∈ rej_Q(t)
           then candidates.remove(ν)
```

Fig. 4. Generic EQA algorithm for a class E of query definitions.

Lemma 1. *For complete d*NWA*s A, the safe states $univ\_sel^A(a,\gamma,P)$ are:*

$$\{p \mid acc^A_{H_{sel}}(p,p_0) \Rightarrow p_0 \in befClose^A(a,\gamma,P)\}$$

Generic Algorithm. Our algorithm will be obtained by instantiating the skeleton in Fig. 4 of a generic EQA algorithm, which is parameterized by a class E of query definitions. The static input of the algorithm is a query definition $e \in E$, and its dynamic input on the stream is a nested word $\mathrm{nw}(t)$. We assume that the nested word is parsed by some parallel preprocessor. Our algorithm can thus process the stream of events of t while knowing their matching relation. It then adds the tuples of $Q(t)$ to the external output collection incrementally at the earliest possible event. The main idea is to generate all candidate tuples, test their aliveness repeatedly, output selected candidates and remove rejected candidates.

*Instantiation for d*NWA*s*[5]. Now suppose that the query is defined by a dNWA A. For every candidate ν we maintain its current state $(p,\mathcal{S}) \in stat^{E(A)}$ and a sequence $\Upsilon \in (\Gamma^{E(A)})^*$ that we call stack, whose length is the depth of the current node of t. Sufficiency for selection $(\nu,\eta) \in sel_Q(t)$ holds iff $p \in \mathcal{S}$.

Updating the current set of candidates at event η means to apply a rule of $E(A)$ to the current state $(p,\mathcal{S}) \in E(A)$, and for opening events to create all new candidates, where the current node is used. More precisely, for generating these candidates, we can restrict to the ones generated by continuations of the runs of old candidates. Thus, the number of candidates to process at an event is bounded by $c+i$, where $c = concur_Q(t)$ is the concurrency of the query Q, and $i = immediate_Q(t)$ is the number of new candidates that immediately get safe for selection or rejection. We have seen already how to apply rules of $E(A)$ in polynomial time in the size of A. The node state of the rule is pushed to stack Υ for opening events, and popped from Υ for closing events.

Theorem 2. *For every d*NWA *A recognizing a canonical language over $\Sigma \times \mathbb{B}^n$, and tree $t \in T_\Sigma$, the time needed to process one event is in $O((c+i)\cdot|\Sigma|\cdot|A|^2)$ and the space in $O(c \cdot d \cdot |A|)$, where $d = depth(t)$ is the depth of t, $c = concur_{Q_A}(t)$ and $i = immediate_{Q_A}(t)$.*

[5] This is an overview without detection of REJECTION SUFFICIENCY. See Appendix D.

Adding Schemas. With respect to sufficiency checking, we can integrate the schema into the query. Validation of the document with respect to the schema is an independent task, that we run in parallel. Given an n-ary query Q and a schema S, define a query Q^S_{sel} with domain T_Σ:

$$Q^S_{sel}(t) = Q(t) \text{ if } t \in S, \text{ and } nod(t)^n \text{ otherwise}$$

It is easy to check that $sel_Q = sel_{Q^S_{sel}}$ so that we can test sufficiency for selection as before. The necessary dNWA constructions are in Appendix E. The overall costs of the resulting EQA algorithm with schemas have already been reported in the introduction.

Conclusion. We distinguished a large class of streamable query-schema definitions defined by dNWAs. This class was obtained by designing an EQA algorithm in a first step and bounding the concurrency of the query and the depth of trees in a second. We have shown that EQA is infeasible for nondeterministic NWAs if $n \geq 1$, as well as for Forward XPath with child and descendant axis. In subsequent work [22], we have shown that these classes are indeed not streamable, and distinguished schema restricted fragments of Forward XPath, that can be proven to be streamable by P-time compilation into dNWAs. This proves that the notion of determinism of dNWAs is essential for streamability. An open question is whether we can extend our EQA algorithm to deterministic pushdown automata.

Acknowledgments. This work was partially supported by the Enumeration project ANR-07-blanc and the project CODEX of ANR-08-Defis.

References

1. Grohe, M., Koch, C., Schweikardt, N.: Tight lower bounds for query processing on streaming and external memory data. TCS **380** (2007) 199–217
2. Schweikardt, N.: Machine models and lower bounds for query processing. In: ACM PODS, (2007) 41–52
3. Bar-Yossef, Z., Fontoura, M., Josifovski, V.: Buffering in query evaluation over XML streams. In: ACM PODS, (2005) 216–227
4. Segoufin, L., Vianu, V.: Validating streaming XML documents. In: ACM PODS. (2002) 53–64
5. Segoufin, L., Sirangelo, C.: Constant-memory validation of streaming XML documents against DTDs. In: ICDT (2007). 299–313
6. Bar-Yossef, Z., Fontoura, M., Josifovski, V.: On the memory requirements of xpath evaluation over xml streams. J. Comput. Syst. Sci. **73** (2007) 391–441
7. Benedikt, M., Jeffrey, A., Ley-Wild, R.: Stream firewalling of XML constraints. In: ACM SIGMOD (2008) 487–498
8. Olteanu, D.: SPEX: Streamed and progressive evaluation of XPath. IEEE Trans. on Know. Data Eng. **19** (2007) 934–949
9. Berlea, A.: Online evaluation of regular tree queries. Nordic Journal of Computing **13** (2006) 1–26

10. Benedikt, M., Jeffrey, A.: Efficient and expressive tree filters. In: FSTTCS. Volume 4855 of LNCS (2007) 461–472
11. Kumar, V., Madhusudan, P., Viswanathan, M.: Visibly pushdown automata for streaming XML. In: WWW, (2007) 1053–1062
12. Miklau, G., Suciu, D.: Containment and equivalence for a fragment of XPath. Journal of the ACM **51** (2004) 2–45
13. Gou, G., Chirkova, R.: Efficient algorithms for evaluating XPath over streams. In: ACM SIGMOD (2007) 269–280
14. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journal of the ACM **56** (2009) 1–43
15. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: 36th ACM Symposium on Theory of Computing (2004) 202–211
16. Seidl, H.: Haskell overloading is DEXPTIME-complete. Information Processing Letters **52** (1994) 57–60
17. Gauwin, O., Niehren, J., Tison, S.: Bounded delay and concurrency for earliest query answering. In: 3rd LATA. Volume 5457 of LNCS. (2009) 350–361
18. Martens, W., Neven, F., Schwentick, T.: Which XML schemas admit 1-pass preorder typing? In: ICDT. (2005) 68–82
19. Madhusudan, P., Viswanathan, M.: Query automata for nested words (2009).
20. Neumann, A., Seidl, H.: Locating matches of tree patterns in forests. In: FSTTCS. (1998) 134–145
21. Gauwin, O., Niehren, J., Roos, Y.: Streaming tree automata. Information Processing Letters **109** (2008) 13–17
22. Gauwin, O.: Streaming Tree Automata and XPath. PhD thesis, Université de Lille 1 (2009)
23. Carme, J., Niehren, J., Tommasi, M.: Querying unranked trees with stepwise tree automata. In: 19th RTA. Volume 3091 of LNCS. (2004) 105–118

A Nested Word Automata

We clarify basic expressiveness and efficiency questions for dNWAs and NWAs by comparison to standard tree automata.

We start with the expressiveness of dNWAs and NWA compared to monadic second-order (MSO) logic. The following variant of Thatcher and Wright's theorem can be found for instance in [14]. The main point is that dNWAs are able to capture the bottom-up determinism of stepwise tree automata [23].

Theorem 3 (Alur & Madhusudan [14]). NWA*s and* d NWA*s capture MSO-definable queries.*

Proof. Stepwise tree automata can converted into NWAs in linear time, such that determinism is preserved. When moving downwards, the NWA ignores all information seen. When moving upwards or left to right, is simulates the stepwise tree automaton. The downwards movement is clearly deterministic, and the other movements remains deterministic when simulating deterministic stepwise tree automata.

Vice versa, it is not difficult to convert NWAs into stepwise tree automata. Which can be determinized into dNWAs. This shows that NWAs, dNWAs, and stepwise tree automata recognize the same tree languages. The variant of Thatcher and Wright's theorem for unranked trees shows that a query Q is MSO-definable iff its canonical language L_Q is recognizable by a (bottom-up) deterministic stepwise tree automaton, which is equivalent to Q being definable by a dNWA.

The polynomial time back-and-fourth translation between stepwise tree automata to NWAs used in the above proof also shows that decision problems of NWAs are equivalent modulo P-time reductions to the corresponding decision problems of standard tree automata on binary trees.

Proposition 5. *Language inclusion and universality are in P-time for* d NWA*s but* EXPTIME*-complete for* NWA*s.*

Proof. A EXPTIME algorithm for universality can be obtained by complementation after determinization and completion, and then emptiness checking. Here, we present a direct proof for EXPTIME-hardness for universality. Let $S \subseteq T_{\{a,b\}}$ be the set of binary trees where a has arity 2 and b arity 0, *i.e.*, $s \in S ::= a(s_1, s_2) \mid b$. A standard tree automaton B recognizing binary trees in S has rules of the form $a(p_1, p_2) \to p$ and $b \to q$ where $p, p_1, p_2 \in \mathit{stat}(B)$. We define NWAs A from tree automata B in linear time such that $L(A) = L(B)$:

$$\begin{array}{l} \mathit{stat}(A) = \mathit{stat}(B) \uplus \{\mathit{ok}\} \\ \mathit{stat}(A) = \Gamma(A) \\ \mathit{fin}(A) = \{\mathit{ok}\} \\ \mathit{init}(A) = \mathit{fin}(B) \end{array} \qquad \frac{b \to p \in \mathit{rul}(B)}{\begin{array}{l} p \xrightarrow{\texttt{op}\ b:\epsilon} \mathit{ok} \in \mathit{rul}(A) \\ \mathit{ok} \xrightarrow{\texttt{cl}\ b:\epsilon} \mathit{ok} \in \mathit{rul}(A) \end{array}} \qquad \frac{a(p_1, p_2) \to p \in \mathit{rul}(B)}{\begin{array}{l} p \xrightarrow{\texttt{op}\ a:p_2} p_1 \in \mathit{rul}(A) \\ \mathit{ok} \xrightarrow{\texttt{cl}\ a:p_2} p_2 \in \mathit{rul}(A) \end{array}}$$

Let C be an NWA that recognizes $T_{\{a,b\}} - S$. An NWA for $L(A) \cup L(C)$ can be constructed in P-time from A. Now, we have $L(B) = S$ if and only if $L(A) \cup L(C) = T_{\{a,b\}}$ so that we can reduce universality of B to universality of A.

$$\langle [child{::}\ell\, F]\rangle_t = \{\pi \mid \exists \pi' \in \langle F\rangle_t.\ (\pi,\pi') \in child^t,\ \ell \in \{*, lab^t(\pi')\}\}$$
$$\langle [//\ell\, F]\rangle_t = \{\pi \mid \exists \pi' \in \langle F\rangle_t.\ (\pi,\pi') \in (child^t)^*,\ \ell \in \{*, lab^t(\pi')\}\}$$
$$\langle [F_1\ and\ F_2]\rangle_t = \langle F_1\rangle_t \cap \langle F_2\rangle_t$$
$$\langle [not\ F]\rangle_t = nod(t) - \langle F\rangle_t$$
$$\langle [true]\rangle_t = nod(t)$$

Fig. 5. Semantics of XPath expressions.

B Complexity of Earliest Query Answering

We study the complexity of selection SUFFICIENCY for various classes E of query definitions, whose expressions $(A, B) \in E$ define a canonical language $L(A)$ and and a schema $L(B)$, and thus a query $Q_{(A,B)}$ of some arity n.

Selection SUFFICIENCY for E obtains as input a pair of $(A, B) \in E$, a tree $t \in L(B)$, an event $\eta \in eve(t)$, and a tuple $\nu \in nod(t)^n$, and returns as outputs the truth value of $(\nu, \eta) \in sel_{Q_A}(t)$. Every EQA algorithm needs to solve the SUFFICIENCY problem, since a node is sufficient at an event if and only if it is the algorithms adds it to the output collection, before this event happens. Let $S_{A,\nu,\eta,t}$ be the set of trees on which ν is selected or that have a prefix different from $t^{\preceq\eta}$.

$$S_{A,\nu,\eta,t} = \{t' \in T_\Sigma \ \mid\ equal_\eta(t, t') \Rightarrow \nu \in Q_A(t')\}$$

Lemma 2. $(\nu, \eta) \in sel_{Q_{(A,B)}}(t) \Leftrightarrow \nu \in nod(t^{\preceq\eta})^n \wedge L(B) \subseteq S_{A,\nu,\eta,t}$.

This reformulation of the definition of $sel_{Q_{A,B}}(t)$ relates SUFFICIENCY to language INCLUSION for classes of Boolean queries. This INCLUSION problem for a class E of Boolean queries inputs an pair $(A, B) \in E$ and outputs the truth value of $L(B) \subseteq L_{Q_A} = L(A)$ since Q_A is Boolean. UNIVERSALITY returns the truth value of $T_\Sigma \subseteq L(A)$ instead.

Lemma 3. *(Hardness) For all classes E of definitions of Boolean queries there is a linear time reduction of* INCLUSION *to* SUFFICIENCY*, and of* UNIVERSALITY *to* SUFFICIENCY *for queries with schema T_Σ.*

Proof. Let $(A, B) \in E$ and $t \in L(B)$ a tree. Since Q_A is Boolean, the definition yield $S_{A,(),\mathtt{start},t} = L(A)$. Thus, Lemma 2 proves that $((), \mathtt{start}) \in sel_{Q_{A,B}}(t)$ if and only if $L(B) \subseteq L(A)$.

We consider Boolean filters in the fragment of Forward XPath mentioned in the core of the paper, where $\ell \in \Sigma \cup \{*\}$:

$$F ::= [child{::}\ell\, F] \ \mid\ [//\ell\, F] \ \mid\ [F_1\ and\ F_2] \ \mid\ [not\ F] \ \mid\ [true]$$

Whether a filter expression is satisfied at the root of a tree t, in formulas $\epsilon \in \langle F\rangle_t$, is recalled in Fig. 5.

Proposition 6. SUFFICIENCY *for Boolean queries defined in the above fragment of Forward XPath is coNP-hard, even without schema assumptions.*

Proof. According to Hardness Lemma 3, SUFFICIENCY without schemas is harder than UNIVERSALITY of Boolean queries. Since Descending XPath provides negation and disjunction in filters, INCLUSION is indeed equivalent to UNIVERSALITY. This holds, since $T_\Sigma \subseteq L_{[[\text{not } \mathrm{F}_1] \text{ or } \mathrm{F}_2]}$ iff $L_{F_1} \subseteq L_{F_2}$. In turn, INCLUSION for Boolean queries in the above fragment of Forward XPath (even without negation and disjunction) was proven coNP-hard by Miklau and Suciu [12].

Proof. The Boolean operations for dNWAs are in P-time, and emptiness checking too. For possibly non-deterministic NWAs, we can encode universality of unrestricted stepwise tree automata, which is EXPTIME-hard, since these can be identified with standard tree automata.

Proposition 7. SUFFICIENCY *for Boolean* NWA *queries is* EXPTIME*-hard, even without schema assumptions.*

Proof. By Lemma 3, SUFFICIENCY without schemas is harder than UNIVERSALITY for NWAs, and thus EXPTIME-hard by Proposition 5.

Lemma 4. *If a d*NWA *A defines a query, we can compute a d*NWA *recognizing language $S_{A,\nu,\eta,t}$ in P-time.*

The crucial point here is that dNWAs can check the equality of the prefixes of two trees until event η deterministically.

Proof. We prove that we can build a dNWA recognizing $S_{A,\nu,\eta,t}$ in polynomial time from A, t, $\pi \in nod(t)$, $\alpha \in \{\texttt{op}, \texttt{cl}\}$, and $\nu \in nod(t)^n$. We define two tree languages:

$$Eq_{t,\eta} = \{t' \mid equal_\eta(t, t')\} \qquad Q_\nu = \{t' \mid \nu \in Q_A(t')\}$$

With these definitions, we get $S_{Q_A,\nu,\eta,t} = Eq_{t,\eta}^{\text{compl}} \cup Q_\nu$ where $L^{\text{compl}} = \{t \in T_\Sigma \mid t \notin L\}$ for $L \subseteq T_\Sigma$. Hence it suffices to build dNWAs recognizing $Eq_{t,\eta}$ and Q_ν in P-time.

First of all, we define a dNWA recognizing $Eq_{t,\eta} = \{t' \mid equal_\eta(t, t')\}$. We set $stat = eve(t^{\preceq\eta})$, $\Gamma = \{\gamma\}$ (arbitrary), $init = \{\texttt{start}\}$, $fin = \{\eta\}$, and the following rules where $\preceq$ and pr are interpreted on $eve(t)$:

$$\frac{(\alpha,\pi) \preceq \eta \qquad a = lab^t(\pi)}{pr((\alpha,\pi)) \xrightarrow{\alpha\ a:\gamma} (\alpha,\pi)} \qquad \frac{a \in \Sigma}{\eta \xrightarrow{\texttt{op}\ a:\gamma} \eta \qquad \eta \xrightarrow{\texttt{cl}\ a:\gamma} \eta}$$

Second, we define a dNWA recognizing the set $Q_\nu = \{t' \mid \nu \in Q_A(t')\}$. Such a dNWA can be built in several steps. We first build a dNWA A' recognizing all trees annotated with the tuple ν, *i.e.*:

$$L(A') = \{t * \nu \mid t \in T_\Sigma\}$$

Then we can intersect A' with A, in order to distinguish all annotated trees on which ν is selected by Q_A. Finally, we can project on the Σ-component in order to obtain the desired trees:

$$Q_\nu = \Pi_\Sigma(Q_A \wedge Q_{A'})$$

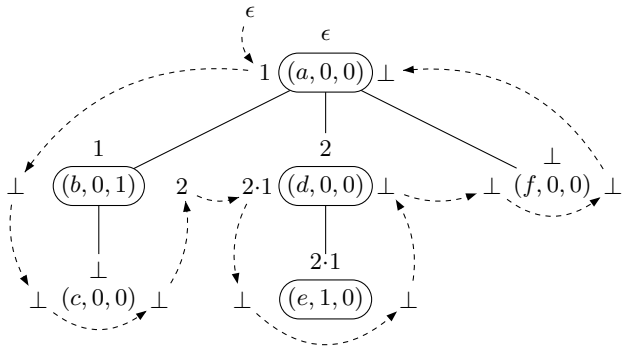


Fig. 6. A run of the dNWA A', when $\nu = (2{\cdot}1, 1)$. The domain for this ν is $domain = \{\epsilon, 1, 2, 2{\cdot}1\}$, as indicated by framed nodes.

The corresponding automata operations preserve determinism, in this particular case: for each tree $t \in T_\Sigma$, there is at most one run of $A \cap A'$ on $t * \nu$, as both automata are deterministic. Hence, after projection, there is also at most one run on t, and thus the determinism is preserved by the projection, in this case.

It remains to detail the construction of A'. If the arity of Q_A is $n = 0$ then $\nu = ()$ and we can take a universal automaton, as $L(A') = T_\Sigma$. Otherwise, in order to define this automaton in polynomial size in $|\nu|$, some preprocessing on ν is required, which factorizes common prefixes of node addresses. Roughly speaking, we call *domain* the domain of the smallest tree containing ν, and build a dNWA that computes in its states the next element of *domain* to be checked, as illustrated in Fig. 6. Formally, let *domain* be the set of positions π smaller or equal to some position of ν for the order defined by $\pi.i < \pi.j$ if $i < j$ and $\pi < \pi.i$. We write $domain_\perp = domain \cup \{\perp\}$. We introduce the function *next*: $\{\mathtt{op}, \mathtt{cl}\} \times (\mathbb{N}^* \cup \{\perp\}) \to domain_\perp$ that indicates whether the domain still continues above (resp. at the right of) the current node π, when called with $(\mathtt{op}, \pi)$ (resp. $(\mathtt{cl}, \pi)$):

$$\begin{cases} next(\mathtt{op}, \pi) & = \pi{\cdot}1 & \text{if } \pi{\cdot}1 \in domain, & \perp \text{ otherwise} \\ next(\mathtt{cl}, \pi{\cdot}i) & = \pi{\cdot}(i+1) & \text{if } \pi{\cdot}(i+1) \in domain, & \perp \text{ otherwise} \\ next(\alpha, \perp) & = \perp & \text{for } \alpha \in \{\mathtt{op}, \mathtt{cl}\} & \end{cases}$$

We also introduce the function $vars_\nu$: $domain_\perp \to \mathbb{B}^n$ that associates with each node the variables corresponding to the annotation by ν:

$$vars_{(\pi_1,\ldots,\pi_n)}(\pi) = (b_1, \ldots, b_n) \text{ where } b_i = \begin{cases} 1 \text{ if } \pi_i = \pi \\ 0 \text{ otherwise} \end{cases}$$

We can now define the dNWA A'. A run of A' is shown in Fig. 6.

$$stat^{A'} = \Gamma^{A'} = domain_\perp \qquad init^{A'} = \epsilon \qquad fin^{A'} = \{\perp\}$$

$$\frac{a \in \Sigma \qquad \pi, \pi' \in domain_\perp \qquad l = vars_\nu(\pi)}{\pi \xrightarrow{\mathtt{op}\ (a,l):\pi} next(\mathtt{op}, \pi) \in rul^{A'} \qquad \pi' \xrightarrow{\mathtt{cl}\ (a,l):\pi} next(\mathtt{cl}, \pi) \in rul^{A'}}$$

$$\frac{p_0 \xrightarrow{\texttt{op}\ a:\gamma_1} p_1 \in rul^A \quad \mathcal{S}_1 = univ\_sel^A(a,\gamma_1,\mathcal{S}_0) \quad \mathcal{R}_1 = univ\_rej^A(a,\gamma_1,\mathcal{R}_0)}{(p_0,\mathcal{S}_0,\mathcal{R}_0) \xrightarrow{\texttt{op}\ a:(\gamma_1,\mathcal{S}_0,\mathcal{R}_0)} (p_1,\mathcal{S}_1,\mathcal{R}_1) \in rul^{E'(A)}}$$

$$\frac{p_0 \xrightarrow{\texttt{cl}\ a:\gamma_0} p_1 \in rul^A \quad \mathcal{S}_0,\mathcal{S}_1,\mathcal{R}_0,\mathcal{R}_1 \subseteq stat^A}{(p_0,\mathcal{S}_0,\mathcal{R}_0) \xrightarrow{\texttt{cl}\ a:(\gamma_0,\mathcal{S}_1,\mathcal{R}_1)} (p_1,\mathcal{S}_1,\mathcal{R}_1) \in rul^{E'(A)}}$$

$$init^{E'(A)} = (init^A,\ fin^A,\ stat^A - fin^A) \qquad fin^{E'(A)} = \{(p,\ fin^A,\ stat^A - fin^A) \mid p \in fin^A\}$$

Fig. 7. Construction of $E'(A)$ from A

Theorem 4. SUFFICIENCY *for* dNWA*s queries and schemas is in polynomial time.*

Proof. We can test $L(B) \subseteq S_{A,\nu,\eta,t}$ in polynomial time, if B is given an dNWA, since we can compute a dNWA for $S_{A,\nu,\eta,t}$ in linear time by Lemma 4, and since INCLUSION for dNWAs is in polynomial time (Proposition 5).

As a corollary SUFFICIENCY for NWAs is EXPTIME-complete. A EXPTIME algorithm follows from NWA determinization and Theorem 4. By Proposition 7, the lower bound holds already for NWAs defining Boolean queries.

C Inferring Safe States

We provide and prove a complete construction, that takes also into account failure states.

The treatment of safe states for rejection is more delicate. Here we have to assume determinism and need a new argument for a proper treatment of partial candidates. The definitions *safe_rej*, *univ_rej* and S_{rej} remain the same, except that we have to replace *sel* by *rej*, $\nu \in nod(t)^n$ by $\nu \in nod_{\oslash}(t)^n$, and H_{sel} by $H_{rej} = T^*_{\Sigma\times\mathbb{B}^n}$. Rejection states at the root are precisely these states: $S_{rej}((\texttt{cl},\epsilon)) = stat^A - fin^A$. Propagation rules remain unchanged too, but correctness requires two new arguments. The critical rule

$$S_{rej}((\texttt{op},\pi)) = univ\_rej^A(a,\gamma,S_{rej}((\texttt{cl},\pi)))$$

remains correct when imposing determinism and completeness on A, since this ensures that a hedge will fail iff a run on this hedge leads to a rejection state. The additional quantification over hedges in H_{rej} (in the definition of *univ_rej*), which may turn continuations into non-canonically annotated trees, doesn't harm, since such trees cannot be recognized by A, when assuming that the language of A is canonical (it defines a query), as we do.

Now the propagation rules allow to infer both $safe\_sel^A_{(\nu,\eta)}(t)$ and $safe\_rej^A_{(\nu,\eta)}(t)$ for all events η. This can be done by running the NWA $E'(A)$ defined in Fig. 7. The signature of $E'(A)$ is still $\Sigma \times \mathbb{B}^n$, as for A. The state sets may also be exponentially large, since $stat^{E'(A)} = stat^A \times 2^{stat^A} \times 2^{stat^A}$ and $\Gamma(E'(A)) =$

$\Gamma(A) \times 2^{stat^A} \times 2^{stat^A}$. Note that E' preserves determinism. Proposition 8 below subsumes Proposition 3.

Proposition 8. *Let A be a dNWA on $\Sigma \times \mathbb{B}^n$ that defines a query, and $t * \nu \in T_{\Sigma \times \mathbb{B}^n}$. Then $E'(A)$ is a dNWA that accepts the same language as A.*

Furthermore, if r^A (resp. $r^{E'(A)}$) is the unique run of A (resp. $E'(A)$) on $t * \nu$ then for all $\eta \in eve(\eta) - \{\mathtt{start}\}$:

$$r^{E'(A)}(\eta) = (r^A(\eta), safe\_sel^A_{(\nu,\eta)}(t), safe\_rej^A_{(\nu,\eta)}(t))$$

Proof. We prove this proposition by Lemmas 5 and 6. For the whole section, we fix A, a dNWA on $\Sigma \times \mathbb{B}^n$ that defines a query, $t * \nu \in T_{\Sigma \times \mathbb{B}^n}$, and we suppose that r^A is the unique run of A on t.

Let us consider the function f that associates a pair $(\mathcal{S}, \mathcal{R}) \in 2^{stat^A} \times 2^{stat^A}$ to each event of $t * \nu$ (except `start`) using the following inference rules:

$$f((\mathtt{cl}, \epsilon)) = (fin^A, stat^A - fin^A) \tag{1}$$

$$\frac{\pi \in nod(t) \qquad f((\mathtt{cl}, \pi)) = (\mathcal{S}, \mathcal{R}) \qquad a = lab^t(\pi) \qquad \gamma = r^A(\pi)}{f((\mathtt{op}, \pi)) = (univ\_sel^A(a, \gamma, \mathcal{S}), univ\_rej^A(a, \gamma, \mathcal{R}))} \tag{2}$$

$$\frac{\pi \in nod(t) \qquad \pi{\cdot}i \in nod(t) \qquad f((\mathtt{op}, \pi)) = (\mathcal{S}, \mathcal{R})}{f((\mathtt{cl}, \pi{\cdot}i)) = (\mathcal{S}, \mathcal{R})} \tag{3}$$

Lemma 5. *For every event $\eta \in eve(t) - \{\mathtt{start}\}$,*

$$f(\eta) = (safe\_sel^A_{(\nu,\eta)}(t), safe\_rej^A_{(\nu,\eta)}(t))$$

Proof. We proceed by induction on events of t (except `start`), according to a top-down, breadth-first, right-to-left traversal of t.

For $(\mathtt{cl}, \epsilon)$, the result is trivial from rule (1) and the definitions of *safe_sel* and *safe_rej*.

Let $e = (\mathtt{op}, \pi)$, and suppose that the property holds for $(\mathtt{cl}, \pi)$. From the application of rule (2), we know that $f(\eta) = (univ\_sel^A(a, \gamma, \mathcal{S}), univ\_rej^A(a, \gamma, \mathcal{R}))$ with $f((\mathtt{cl}, \pi)) = (\mathcal{S}, \mathcal{R})$, $a = lab^t(\pi)$ and $\gamma = r^A(\pi)$. By definition, we have: $univ\_sel^A(a, \gamma, \mathcal{S}) = \{p \mid \forall h \in H_{sel}.\ ev\_cl^A(h, p, a, \gamma) \in \mathcal{S}\}$, and by induction hypothesis, $\mathcal{S} = safe\_sel^A_{(\nu,(\mathtt{cl},\pi))}(t)$.

We first prove that $univ\_sel^A(a, \gamma, \mathcal{S}) = safe\_sel^A_{(\nu,\eta)}(t)$. Suppose that $p \in safe\_sel^A_{(\nu,\eta)}(t)$. Let $h \in H_{sel}$, and $p' = ev\_cl^A(h, p, a, \gamma)$. Then $p' \in safe\_sel^A_{(\nu,(\mathtt{cl},\pi))}(t)$, as sufficiency remains true for events following η. Thus, $p \in univ\_sel^A(a, \gamma, \mathcal{S})$. Conversely, if $p \in univ\_sel^A(a, \gamma, \mathcal{S})$ then $\nu \in nod(t^{\preceq \eta})^n$ (consider the empty continuation). So for every $t' \in T_\Sigma$ such that $equal_\eta(t, t')$, the hedge h of children of π in t' is in H_{sel}. Thus $ev\_cl^A(h, p, a, \gamma) \in safe\_sel^A_{(\nu,(\mathtt{cl},\pi))}(t)$, which means that $\nu \in Q_A(t')$, so η is sufficient for selecting ν, and $p \in safe\_sel^A_{(\nu,\eta)}(t)$. Finally, $univ\_sel^A(a, \gamma, \mathcal{S}) = safe\_sel^A_{(\nu,\eta)}(t)$.

Now we prove the similar result for safe states for rejection, *i.e.*, that:

$$univ\_rej^A(a,\gamma,\mathcal{R}) = safe\_rej^A_{(\nu,\eta)}(t)$$

The difference here is that we deal with partial candidates. We write $\nu^{\preceq\eta}$ for the partial tuple obtained by replacing every component strictly after η by $\oslash$. Inclusion $safe\_rej^A_{(\nu,\eta)}(t) \subseteq univ\_rej^A(a,\gamma,\mathcal{S})$ holds for the same reason, namely events following η remain sufficient for failing, even for completions of $\nu^{\preceq\eta}$. Now suppose that $p \in univ\_rej^A(a,\gamma,\mathcal{S})$. Fix $t' \in T_\Sigma$ such that $equal_\eta(t,t')$, and let h be the hedge of children of π in t'. Then $ev\_cl^A(h,p,a,\gamma) \in safe\_rej^A_{(\nu,(\mathtt{cl},\pi))}(t)$, and thus every completion ν' of $\nu^{\preceq\eta}$ after η fails. Hence η is sufficient for failing $\nu^{\preceq\eta}$, and $p \in safe\_rej^A_{(\nu,\eta)}(t)$.

Finally we consider $e = (\mathtt{cl},\pi{\cdot}i)$, and assume that the property holds for $(\mathtt{op},\pi)$ and $(\mathtt{cl},\pi)$. From Rule (3) and induction hypothesis, we obtain that: $f((\mathtt{cl},\pi{\cdot}i)) = (safe\_sel^A_{(\nu,(\mathtt{op},\pi))}(t), safe\_rej^A_{(\nu,(\mathtt{op},\pi))}(t))$.

First we prove that $safe\_sel^A_{(\nu,(\mathtt{op},\pi))}(t) = safe\_sel^A_{(\nu,\eta)}(t)$. We have:

$p \in safe\_sel^A_{(\nu,(\mathtt{op},\pi))}(t)$
$\Leftrightarrow (\exists r \in p\_runs^A(t*\nu) \wedge r((\mathtt{op},\pi)) = p) \Rightarrow (\nu,(\mathtt{op},\pi)) \in sel_{Q_A}(t)$
$\Leftrightarrow (\exists r \in p\_runs^A(t*\nu) \wedge r((\mathtt{op},\pi)) = p) \Rightarrow$
$\quad \forall h \in H_{sel}.\ ev\_cl^A(h,p,a,\gamma) \in safe\_sel^A_{(\nu,(\mathtt{cl},\pi))}(t)$
$\Leftrightarrow (\exists r \in p\_runs^A(t*\nu) \wedge r((\mathtt{cl},\pi{\cdot}i)) = p) \Rightarrow (\nu,(\mathtt{cl},\pi{\cdot}i)) \in sel_{Q_A}(t)$
$\Leftrightarrow p \in safe\_sel^A_{(\nu,(\mathtt{cl},\pi{\cdot}i))}(t)$

We now show that $safe\_rej^A_{(\nu,(\mathtt{op},\pi))}(t) = safe\_rej^A_{(\nu,\eta)}(t)$:

$p \in safe\_rej^A_{(\nu,(\mathtt{op},\pi))}(t)$
$\Leftrightarrow (\exists r \in p\_runs^A(t*\nu) \wedge r((\mathtt{op},\pi)) = p) \Rightarrow (\nu,(\mathtt{op},\pi)) \in rej_{Q_A}(t)$
$\Leftrightarrow (\exists r \in p\_runs^A(t*\nu) \wedge r((\mathtt{op},\pi)) = p) \Rightarrow$
$\quad \forall h \in H_{rej}.ev\_cl^A(h,p,a,\gamma) \in safe\_rej^A_{(\nu_h,(\mathtt{cl},\pi))}(t)$
$\Leftrightarrow (\exists r \in p\_runs^A(t*\nu) \wedge r((\mathtt{cl},\pi{\cdot}i)) = p) \Rightarrow (\nu,(\mathtt{cl},\pi{\cdot}i)) \in rej_{Q_A}(t)$
$\Leftrightarrow p \in safe\_rej^A_{(\nu,(\mathtt{cl},\pi{\cdot}i))}(t)$

where ν_h is obtained from ν by adding annotations of h.

Lemma 6. *There is a run $(r^{E'(A)}, r^{E'(A)})$ of $E'(A)$ on $t*\nu$, and for every event $\eta \in eve(t) - \{\mathtt{start}\}$,*

$$r^{E'(A)}(\eta) = (r^A(\eta),\mathcal{S},\mathcal{R}) \text{ with } (\mathcal{S},\mathcal{R}) = f(\eta)$$

Proof. Inference schemas defining $E'(A)$ show that every run r of A has a unique corresponding run r' in $E'(A)$, and r is the first component of r'.

Again, we use an induction on events of t (except `start`) according to a top-down, breadth-first, left-to-right traversal of t.

For $e = (\mathtt{cl},\epsilon)$, we have $f(\eta) = (fin^A, stat^A - fin^A)$. At the root, we have $r(\epsilon) = (r^A(\epsilon), fin^A, stat^A - fin^A)$, so $r((\mathtt{cl},\epsilon)) = (r^A((\mathtt{cl},\epsilon)), fin^A, stat^A - fin^A)$.

Now consider that $e = (\mathtt{op},\pi)$ and suppose that we have $r^{E'(A)}((\mathtt{cl},\pi)) = (r^A((\mathtt{cl},\pi)),\mathcal{S}',\mathcal{R}')$ with $(\mathcal{S}',\mathcal{R}') = f((\mathtt{cl},\pi))$. This implies that $r^{E'(A)}(\pi) =$

$$\frac{a \in \Sigma \times \mathbb{B}^n \quad p_1 \xrightarrow{\texttt{op}\ a:\gamma} p_3 \in rul^A \quad p_4 \xrightarrow{\texttt{cl}\ a:\gamma} p_2 \in rul^A}{acc_{H_{rej}}(p_1, p_2) \text{ :- } acc_{H_{rej}}(p_3, p_4).}$$

$$\frac{p \in stat^A}{acc_{H_{rej}}(p, p).} \qquad \frac{p_1, p_2, p_3 \in stat^A}{acc_{H_{rej}}(p_1, p_2) \text{ :- } acc_{H_{rej}}(p_1, p_3), acc_{H_{rej}}(p_3, p_2).}$$

Fig. 8. Inference rules for the definition of $acc^A_{H_{rej}}$

$(r^A(\pi), \mathcal{S}', \mathcal{R}')$, so we get $\mathcal{S}' = univ\_sel^A(a, \gamma, \mathcal{S})$, $\mathcal{R}' = univ\_rej^A(a, \gamma, \mathcal{R})$ and $r^{E'(A)}(\eta) = (r^A(\eta), \mathcal{S}, \mathcal{R})$ where $a = lab^t(\pi)$ and $\gamma = r^A(\pi)$. Hence, $(\mathcal{S}, \mathcal{R}) = f((\texttt{op}, \pi))$.

Finally, let us assume that $e = (\texttt{cl}, \pi{\cdot}i)$ and also that $r^{E'(A)}((\texttt{op}, \pi)) = (r^A((\texttt{op}, \pi)), \mathcal{S}, \mathcal{R})$ with $\mathcal{S}, \mathcal{R}$ defined by $(\mathcal{S}, \mathcal{R}) = f((\texttt{op}, \pi))$. By an immediate induction on children of π, each child $\pi{\cdot}j$ of π verifies $r^{E'(A)}(\pi{\cdot}j) = (r^A(\pi{\cdot}j), \mathcal{S}, \mathcal{R})$ and for the state $r^{E'(A)}((\texttt{cl}, \pi{\cdot}j)) = (r^A((\texttt{cl}, \pi{\cdot}j)), \mathcal{S}, \mathcal{R})$, and in particular for $j = i$. From rule (3) of the definition of f, we know that $(\mathcal{S}, \mathcal{R}) = f((\texttt{cl}, \pi{\cdot}i))$.

D EQA Algorithm for dNWAs

To build $E'(A)$ on-the-fly, one need to compute $univ\_rej(a, \gamma, P)$ at each opening event. To do so, we use the accessibility relation $acc^A_{H_{rej}}$ of A through hedges of H_{rej}.

The following proposition subsumes Proposition 4.

Proposition 9. *The collection of values $acc^A_{H_X}(p_1, p_2)$ with $X \in \{sel, rej\}$ and $p_1, p_2 \in stat^A$ can be computed in time $O(|\Sigma| \cdot |A|^3)$ for every d*NWA *A.*

Proof. The completion of the dNWA A is done in time $O(|\Sigma| \cdot |\Gamma^A| \cdot |stat^A|)$. This bound is also a bound on the number of rules of A_c, the completed automaton. The number of rules of the ground Datalog programs defining $acc_{H_{sel}}$ and $acc_{H_{rej}}$ (see Fig. 3) are in $O(|stat^{A_c}|^3 + |rul^{A_c}| \cdot |stat^{A_c}|)$. Then the saturation process of a ground Datalog program is in linear time.

The following lemma subsumes Lemma 1.

Lemma 7. *For deterministic and complete A, the safe states $univ\_X^A(a, \gamma, P)$ are equal to:*

$$\{p \mid acc^A_{H_X}(p, p_0) \Rightarrow p_0 \in befClose^A(a, \gamma, P)\}$$

*Instantiation for d*NWA*s.* Here we have to maintain the current state $(p, \mathcal{S}, \mathcal{R}) \in stat^{E'(A)}$ and a sequence $\Upsilon \in (\Gamma^{E'(A)})^*$. Sufficiency for selection $(\nu, e) \in sel_Q(t)$ is verified by testing $p \in \mathcal{S}$, and sufficiency for rejection $(\nu, e) \in rej_Q(t)$ by checking $p \in \mathcal{R}$. Updating the current state is done by applying a rule of $E'(A)$, that we can compute using the alternative definition of $univ\_X$ above.

Proposition 2 (as recalled below) remains true when detection of rejection sufficiency is added. Here we use $\mathit{immediate}(t, Q, S)$ to measure the number of new candidates for which sufficiency for selection or rejection can be decided immediately. For a tuple ν and a node π, we write $\nu - \pi$ for the tuple obtained from ν by replacing π by $\oslash$.

$$
\begin{aligned}
&\mathit{immediate}(t, Q, S)\\
&\quad = \max_{\pi \in nod(t)} |\{\nu \mid \begin{cases} \nu - \pi \neq \nu \wedge \\ \nu - \pi \quad \text{is alive at event} \quad pr((\mathtt{op}, \pi)) \wedge \} | \\ \quad \nu \quad \text{is not alive at event} \quad (\mathtt{op}, \pi) \end{cases}\\
&\quad = \max_{\pi \in nod(t)} |\{\nu \mid \begin{cases} \nu - \pi \neq \nu \wedge \\ (\nu - \pi, pr((\mathtt{op}, \pi))) \notin sel^S_Q(t) \cup rej^S_Q(t) \wedge \} | \\ (\quad \nu \quad , \quad (\mathtt{op}, \pi) \quad) \in sel^S_Q(t) \cup rej^S_Q(t) \end{cases}
\end{aligned}
$$

Proposition 2. *For every d*NWA A *recognizing a canonical language over* $\Sigma \times \mathbb{B}^n$*, and tree* $t \in T_\Sigma$*, the time needed to process one event is in* $O((c+i) \cdot |\Sigma| \cdot |A|^2)$ *and the space in* $O(c \cdot d \cdot |A|)$*, where* $d = \mathit{depth}(t)$ *,* $c = \mathit{concur}_{Q_A}(t)$ *and* $i = \mathit{immediate}_{Q_A}(t)$*.*

Proof. Processing an opening event requires more computations than a closing one, as it needs to determine the sufficient events.

Given a label $a \in \Sigma$ and a current state $(p_0, \mathcal{S}_0, \mathcal{R}_0)$ for the partial run of the candidate, there are at most 2^n rules of A_c of the form $p_0 \xrightarrow{\mathtt{op}\ a_\beta : \gamma_1} p_1$. For each of these rules, the computation of $\mathit{befClose}(a_\beta, \gamma_1, \mathcal{S}_0)$ can be performed in time $O(|rul^{A_c}|) = O(|\Sigma| \cdot |stat^A|^2)$ (see proof of Proposition 4). Then, the computation of $\mathit{univ\_X}$ where $X \in \{sel, rej\}$ can be done in time $O(|stat^A|^2)$.

Space consumption is due to the computation of sets of safe states and their storage inside the stack.

Implementation. We provide here a more precise and efficient procedure for the computation of safe states $\mathit{univ\_X}$ where $X \in \{sel, rej\}$ for a dNWA A. We first exhibit some properties of the function mapping sets P to $\mathit{befClose}(a, \gamma, P)$ and, where a and γ are fixed.

Lemma 8. *For every* $a \in \Sigma$*,* $\gamma \in \Gamma^A$*, and* $P_1, P_2 \subseteq stat^A$*:*

$$\mathit{befClose}(a, \gamma, P_1 \cup P_2) = \mathit{befClose}(a, \gamma, P_1) \cup \mathit{befClose}(a, \gamma, P_2)$$

So we get $\mathit{befClose}(a, \gamma, P_2) = \cup_{p \in P_2} \mathit{befClose}(a, \gamma, \{p\})$. Hence we can precompute $\mathit{befClose}(a, \gamma, \{p\})$ for each $a \in \Sigma$, $\gamma \in \Gamma^A$ and $p \in stat^A$, and reuse it for computing $\mathit{befClose}(a, \gamma, P_2)$. This precomputation requires time $O(|\Sigma| \cdot |A|^3)$ and space $O(|\Sigma| \cdot |A|^2)$. An alternative can be not to precompute, but to compute on-the-fly, and keep in memory the results.

Now we look into more details the properties of the function mapping sets P to $\mathit{univ\_X}(a, \gamma, P)$ for fixed a and γ.

Lemma 9. *For every* $a \in \Sigma$*,* $\gamma \in \Gamma^A$*,* $P_1, P_2 \subseteq stat^A$ *and* $X \in \{sel, rej\}$*:*

$$\mathit{univ\_X}(a, \gamma, P_1 \cup P_2) \supseteq \mathit{univ\_X}(a, \gamma, P_1) \cup \mathit{univ\_X}(a, \gamma, P_2)$$

A consequence is that the function mapping sets $P \rightarrow univ\_X(a,\gamma,P)$ is monotonic. Note that in the general case, $univ\_X(a,\gamma,P_1 \cup P_2) \not\subseteq univ\_X(a,\gamma,P_1) \cup univ\_X(a,\gamma,P_2)$. A counter example will be given later on in Appendix F. There we have $(0,2) \notin univ\_rej(a_1,0,\{(1,2)\})$ and $(0,2) \notin univ\_rej(a_1,0,\{(3,2)\})$, but $(0,2) \in univ\_rej(a_1,0,\{(1,2),(3,2)\})$.

```
fun univ_X(a,γ,P)
  let safeStates = set.new(∅)
  let beforeCl = ∪_{p∈P} befClose(a,γ,{p})
  let agenda = beforeCl
in

  // first we set the agenda to what really needs to be computed
  for P1 ⊆ P such that univ_X(a,γ,P1) is memorized
    let U = univ_X(a,γ,P1)
    in
      safeStates.add(U)
      agenda.remove(U)

  // then we perform the needed computations
  for p in agenda
    is_safe = true
    for p' such that acc_{H_X}(p,p')
      if p' not in beforeCl
        is_safe = false
    if is_safe
      safeStates.add(p)

  return safeStates
```

Fig. 9. Algorithm computing $univ\_X(a,\gamma,P)$

Algorithm of Fig. 9 uses these results, and also the fact that, from Lemma 7, $univ\_X(a,\gamma,P_2) \subseteq befClose(a,\gamma,P_2)$. Note that if we choose to store all the computations of safe states (used in the first *for* loop), this can use memory of size $O(|\Sigma| \cdot |\Gamma^A| \cdot |2^{stat^A}|^2)$. However, this can be weakened. For instance a good trade-off between memory and time consumption can be to store all safe states of all previous siblings of the current branch. The reason is that the safe states at opening are computed from the safe states at closing, which are the same for all siblings. Thus, if two siblings have the same label and the same associated stack symbols, their safe states are equal.

E Adding Schemas

In the core of the paper, we introduced the query Q^S_{sel}, that was used for computing selection sufficiency for both the query Q and schema S: $sel^S_Q = sel_{Q^S_{sel}}$.

As for selection, we can integrate the schema into the query such that detecting rejection on the query Q with schema S is equivalent to detecting rejection on the new query Q^S_{rej}. In other terms, $rej^S_Q = rej_{Q^S_{rej}}$ when Q^S_{rej} is defined by:

$$Q^S_{rej}(t) = Q(t) \text{ if } t \in S, \text{ and } \emptyset \text{ otherwise}$$

$$\frac{p_0 \xrightarrow{\texttt{op}\ a_\beta:\gamma_1} p_1 \in rul^A \qquad p'_0 \xrightarrow{\texttt{op}\ a:\gamma'_1} p'_1 \in rul^B}{(p_0,p'_0) \xrightarrow{\texttt{op}\ a_\beta:(\gamma_1,\gamma'_1)} (p_1,p'_1)\ \in rul^{B_{sel}}}$$

$$\frac{p_0 \xrightarrow{\texttt{cl}\ a_\beta:\gamma_0} p_1 \in rul^A \qquad p'_0 \xrightarrow{\texttt{cl}\ a:\gamma'_0} p'_1 \in rul^B}{(p_0,p'_0) \xrightarrow{\texttt{cl}\ a_\beta:(\gamma_0,\gamma'_0)} (p_1,p'_1)\ \in rul^{B_{sel}}}$$

$$init^{B_{sel}} = init^A \times init^B \qquad fin^{B_{sel}} = (fin^A \times fin^B)\ \cup\ (stat^A \times (stat^B - fin^B))$$

Fig. 10. Construction of B_{sel} from A and B

For selection detection, the idea is to build an automaton B_{sel} from the NWAs A and B recognizing respectively the canonical language of Q and the schema S, where B_{sel} is such that $Q_{B_{sel}} = Q^S_{sel}$. This automaton will be similar to the product automaton of A and B, but final states will be enriched by all invalid selections, as introduced in the definition of Q^S_{sel}. Fig. 10 shows how to obtain the NWA B_{sel}. Prior to this construction, A and B must be determinized and completed. For failure detection, we proceed the same way to obtain B_{rej} such that $Q_{B_{rej}} = Q^S_{rej}$. The only difference between B_{sel} and B_{rej} lies in the final states: $fin^{B_{rej}} = fin^A \times fin^B$.

This way, we can compute the safe states for selection with $E'(B_{sel})$ and the safe states for failure with $E'(B_{rej})$. From an implementation point of view, there is no need to compute the safe states for failure of $E'(B_{sel})$ and the safe states for selection of $E'(B_{rej})$. Thus, we can run the efficient algorithm presented in Section 6 and compute the same amount of safe states as for $E'(A)$, but on a bigger automaton. A complete example is provided in Appendix F.

B_{sel} is similar to a product automaton between A_c and B_c, the automata obtained after completion of A and B. Its number of rules and the time to compute it are both in $O(|rul^{A_c}| \cdot |rul^{B_c}|)$. The combination of Propositions 4 and 2 yields our main result:

Theorem 5. *Our* EQA *algorithm for answering queries $Q_{(A,B)}$ defined by a d*NWA*s recognizing canonical language and schema of the query, applied to a tree $t \in L(B)$ on the input stream, has the following costs:*

- *precomputation in time $O(|\Sigma| \cdot |A|^3 \cdot |B|^3)$*
- *time per step in $O((c+i) \cdot |\Sigma| \cdot |A|^2 \cdot |B|^2)$ where $c = concur_{Q_{(A,B)}}(t)$ and $i = immediate_{Q_{(A,B)}}(t)$*
- *space per step in $O(c \cdot d \cdot |A| \cdot |B|)$, where $d = depth(t)$ is the depth of t*

F Example Run of Algorithm with Schema

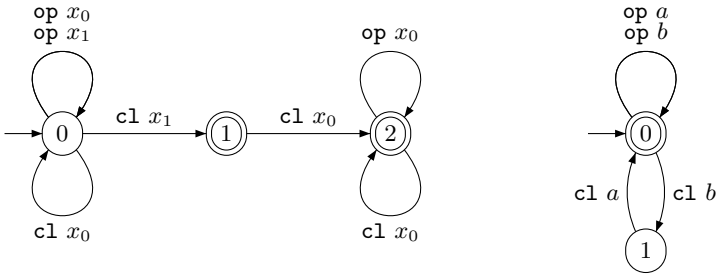
For illustration consider the monadic query Q_0 that selects all nodes without next sibling. It can be defined in MSO by the formula $\neg\exists y.ns(x,y)$. The root of t is selected, and this can be decided when opening it. Without schema, membership

$\pi \in Q_0(t)$ cannot always be decided at opening time, so the algorithm needs to memorize nodes until, either encountering the opening event of the next sibling (for nodes $\pi \notin Q_0(t)$) or the closing the father (for selected nodes $\pi \in Q_0(t)$). When assuming the DTD $a \to (a^*b)^*$ and $b \to \epsilon$, one knows that all a-nodes except the root have a next sibling in all trees satisfying the DTD, so selection of a nodes be decided early at opening time. For b-nodes, selection can still be decided only later, when closing the parent.

We consider the schema S_0 which corresponds to the DTD $\{a \to a^*b,\ b \to \epsilon\}$. We show how the algorithm would behave on this input.

In the sequel, we write a_β for the annotated node label $(a, \beta) \in \Sigma \times \mathbb{B}^n$.

In the following figures, we omit stack symbols as only one occurs in each automaton. Moreover, whenever x occurs in a rule, this means that this rules exists for $x \in \{a, b\}$. Let A be the NWA represented in Fig. 11(a), and B the NWA of Fig. 11(b). We have $Q_0 = Q_A$ and $S_0 = L(B)$.



(a) NWA A recognizing L_{Q_0} (b) NWA B for DTD $\{a \to a^*b,\ b \to \epsilon\}$

Fig. 11. Input NWAs

In the following, we show how safe states are computed in the case with schema, as explained in Section 6 and Appendix E. We start by completing A with the sink state 3 and B with the sink state 2. By applying the inference rules of Fig. 10, we obtain the NWA B_{sel} represented in Fig. 12 (states resulting from completion are omitted for clarity). The NWA B_{rej} only differs on final states.

Then we compute the relations $acc_{H_{sel}}$ and $acc_{H_{rej}}$. Fig. 13 is an array of Booleans representing the relation $acc_{H_{rej}}$. States (p_0, p_1) are written p_0p_1 for sake of conciseness. The relation $acc_{H_{sel}}$ is obtained from this array by replacing values in italics by 0.

Suppose that we want to compute the safe states at the root for the labeling a_0 on our example. This corresponds to computing $\mathit{safe\_sel}^{B_{sel}}(a_0, \gamma, \mathit{fin}^{B_{sel}})$. First, we obtain from the "cl" rules of B_{sel}:

$$\mathit{befClose}^{B_{sel}}(a_0, \gamma, \mathit{fin}^{B_{sel}}) = \{(0,0),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2),(3,0),(3,2)\}$$

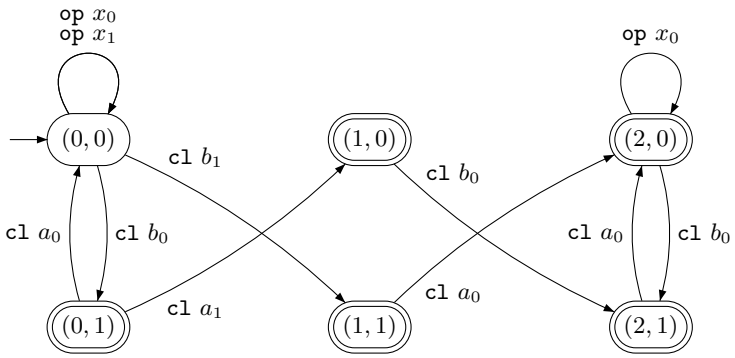


Fig. 12. NWA B_{sel} obtained from A and B (sink states are omitted)

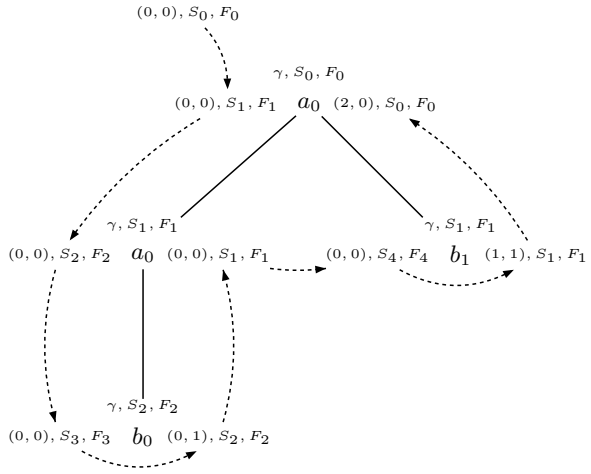
$acc_{H_{rej}}$	00	01	02	10	11	12	20	21	22	30	31	32
00	1	1	1	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
01	0	1	1	0	0	*1*	0	0	*1*	0	0	*1*
02	0	0	1	0	0	*1*	0	0	*1*	0	0	*1*
10	0	0	0	1	0	0	0	0	0	1	1	1
11	0	0	0	0	1	0	0	0	0	0	0	1
12	0	0	0	0	0	1	0	0	0	0	0	1
20	0	0	0	0	0	0	1	1	1	*1*	*1*	*1*
21	0	0	0	0	0	0	0	1	1	0	0	*1*
22	0	0	0	0	0	0	0	0	1	0	0	*1*
30	0	0	0	0	0	0	0	0	0	1	1	1
31	0	0	0	0	0	0	0	0	0	0	1	1
32	0	0	0	0	0	0	0	0	0	0	0	1

Fig. 13. $acc_{H_{rej}}$ associated to Q_0 and S_0

We denote this set BC_1. From the previous section, we can look at which states p verify $acc_{H_{sel}}(p) \subseteq BC_1$. These states are the safe states:

$$safe\_sel^{B_{sel}}(a_0, \gamma, fin^{B_{sel}}) = \{(0,2),(1,1),(1,2),(2,0),(2,1),(2,2),(3,2)\}$$

Using this processing at each opening event for safe states and failure states, we obtain the run on the canonical tree $a_0(a_0(b_0), b_1)$ represented in Fig. 14. Here, safe states $\mathcal{S}$ are those provided by B_{sel} and failure states $\mathcal{R}$ are those provided by B_{rej}. We only represent them as they are the only relevant ones (failure states computed by B_{sel} are useless, for instance).



(a) Run of the algorithm on a tree for one candidate

S_0={(0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2), (3, 1), (3, 2)}
S_1=S_2={(0, 2), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2), (3, 2)}
S_3={(0, 1), (0, 2), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2), (3, 1), (3, 2)}
S_4={(0, 0), (0, 1), (0, 2), (1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2)}
F_0={(0, 0), (0, 1), (0, 2), (1, 1), (1, 2), (2, 1), (2, 2), (3, 0), (3, 1), (3, 2)}
F_1={(0, 1), (0, 2), (1, 0), (1, 2), (2, 2), (3, 0), (3, 1), (3, 2)}
F_2={(0, 2), (1, 0), (1, 2), (2, 2), (3, 0), (3, 1), (3, 2)}
F_3={(0, 1), (0, 2), (1, 1), (1, 2), (2, 1), (2, 2), (3, 0), (3, 1), (3, 2)}
F_4={(0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2), (3, 0), (3, 1), (3, 2)}

(b) Sets involved in this run

Fig. 14. Run of the algorithm on a tree